Um Mecanismo para Coleta Automatizada de Evidências Digitais em Honeypots de Alta Interatividade

Martim d'Orey Posser de Andrade Carbone[1]*, **Paulo Lício de Geus**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas
Caixa Postal 6176 – 13083-970 Campinas, SP

{martim,paulo}@las.ic.unicamp.br

Abstract. *Honeypots are computational resources whose value resides in being probed, attacked or compromised by invaders. This makes it possible to obtain information about their methods, tools and motivations. In high-interaction honeypots, this is done, among other ways, by collecting digital evidence. This collection is traditionally done manually and statically, demanding time and not always generating good results. In this paper, we describe an automatic, dynamic and transparent mechanism based on system call interception for collecting digital evidence on honeypots, filling in the flaws found in the traditional methodologies.*

Resumo. *Honeypots são recursos computacionais cujo valor está em sua sondagem, ataque ou comprometimento por invasores. Com isso, é possível obter informações a respeito de seus métodos, ferramentas e motivações. Em honeypots de alta interação, esta obtenção se dá, principalmente, através da coleta de evidências digitais, que é tradicionalmente feita manualmente e estaticamente, exigindo tempo e nem sempre rendendo bons resultados. Neste artigo, é descrito um mecanismo automático, dinâmico e transparente baseado na interceptação de chamadas de sistema para a coleta de evidências digitais em honeypots, eliminando as falhas encontradas nas metodologias tradicionais.*

1. Introdução

Desde que foi inicialmente proposto no início dos anos 90 [Stoll, 1991, Cheswick, 1992], o conceito de *honeypot* sofreu uma grande popularização na comunidade de segurança da informação, sendo atualmente reconhecido como uma ferramenta potencialmente valiosa no combate aos invasores [The Honeynet Project, 2001, Spitzner, 2002].

Um *honeypot* é definido como um recurso computacional cujo valor está em sua sondagem, ataque ou comprometimento por invasores [Spitzner, 2003a]. Esta definição é bastante ampla, podendo abranger desde um complexo ambiente computacional até simples arquivos [Spitzner, 2003b]. Neste artigo, não obstante, consideraremos *honeypots* como sendo computadores físicos. O valor de um *honeypot* está diretamente ligado ao fato de o recurso não ser utilizado por usuários legítimos na realização de atividades de produção do dia-dia. Isso torna toda e qualquer atividade envolvendo o *honeypot* naturalmente suspeita, praticamente eliminando a ocorrência de falsos positivos.

Esta idéia, apesar de simples, é extremamente poderosa, podendo ser utilizada para diversos fins: captura de assinaturas de ataque, estudo dos hábitos e motivações dos invasores, estudo

*Apoio FAPESP

de ferramentas maliciosas, entre outras aplicações [Spitzner, 2002, The Honeynet Project, 2001, Spitzner, 2003b].

Da mesma forma que um *honeypot* pode possuir diferentes objetivos, há também diversos tipos de *honeypots*, adequados a diferentes situações e diferentes tipos de dados que se pode desejar coletar [Spitzner, 2002]. *Honeypots* de *baixa interatividade* possuem um baixo nível de interatividade com o atacante, limitando suas ações [Provos, 2003]. Um exemplo clássico consiste em uma máquina executando um monitorador de portas, com o objetivo de detectar e registrar ataques remotos. Aqui, a interação entre o atacante e o *honeypot* fica restrita à pilha TCP/IP.

No outro extremo, há os *honeypots* de *alta interatividade*, com os quais o invasor pode interagir totalmente, como se estivesse trabalhando em sua própria máquina. Não é difícil notar que esta segunda abordagem possibilita a obtenção de uma quantidade muito maior de dados se comparada com a primeira, como resultado de uma maior interação entre o invasor e o *honeypot* [Spitzner, 2002]. Estes dados podem possuir origens diversas e naturezas distintas: tráfego de rede capturado, *logs* do sistema operacional, comandos digitados pelo invasor, conteúdo da memória volátil, alterações no sistema de arquivos, entre outros. Tais dados constituem exemplos de *evidências digitais* de uma intrusão, objeto de estudo em metodologias de forense computacional [dos Reis, 2003] e a principal fonte de aprendizado quando se trata de *honeypots*.

A extração de evidências digitais não é um processo trivial [dos Reis, 2003], e esta dificuldade ganha outra dimensão quando se trabalha com *honeypots*. Neste último caso, torna-se necessário não apenas criar métodos eficazes para extrair as evidências, mas também fazer com que estes métodos não despertem suspeitas por parte do invasor, que deve acreditar que está invadindo uma máquina de produção legítima.

Certos tipos de evidências digitais, no entanto, podem ser obtidos de forma mais transparente que outros. O tráfego de rede oriundo de um invasor, por exemplo, pode ser obtido passivamente através de um capturador de pacotes, sem que se corra o risco de alertá-lo. Há outras evidências, no entanto, cuja extração é mais delicada, e exige maiores cuidados por parte do administrador do *honeypot*. Estas evidências normalmente estão localizadas em seu interior, em locais como a memória RAM e o sistema de arquivos. Este último é considerado a principal fonte de evidências digitais em um sistema [dos Reis, 2003], e merece atenção especial no estudo de *honeypots*. Este artigo se focará nas evidências digitais deixadas em sistemas de arquivos de *honeypots* executando o sistema operacional Linux com *kernel* versão 2.4.

Tradicionalmente, a extração de evidências do sistema de arquivos de um *honeypot* é realizada de forma estática e pouco automatizada, exigindo participação ativa do administrador [The Honeynet Project, 2001, Equipe Honeynet.BR, 2002]. A técnica mais comum consiste na criação de uma imagem estática do sistema de arquivos, através de um aplicativo de cópia de dados, como o utilitário *dd* do Linux. Esta imagem é então transportada a uma máquina segura, onde é submetida a uma análise forense. Nesta análise, é comum o uso de verificadores de integridade de sistemas de arquivos, como o *tripwire*, entre outras ferramentas. O objetivo é detectar alterações no sistema de arquivos analisado com relação ao original, com isso determinando as modificações realizadas por um invasor.

Este método possui uma série de desvantagens. Além de ser manual e lento, em muitos casos ele exige que o *honeypot* seja temporariamente desativado para que seja feita a criação da imagem (que geralmente envolve a cópia de um grande volume de dados) ou mesmo desligado. Esta exigência tem conseqüências negativas na quantidade de evidências coletadas, além de potencialmente alertar invasores para a real função da máquina. Como se não bastasse, este método atua de forma estática, ou seja, quaisquer evidências cujo tempo de vida esteja situado entre duas checagens não serão detectadas.

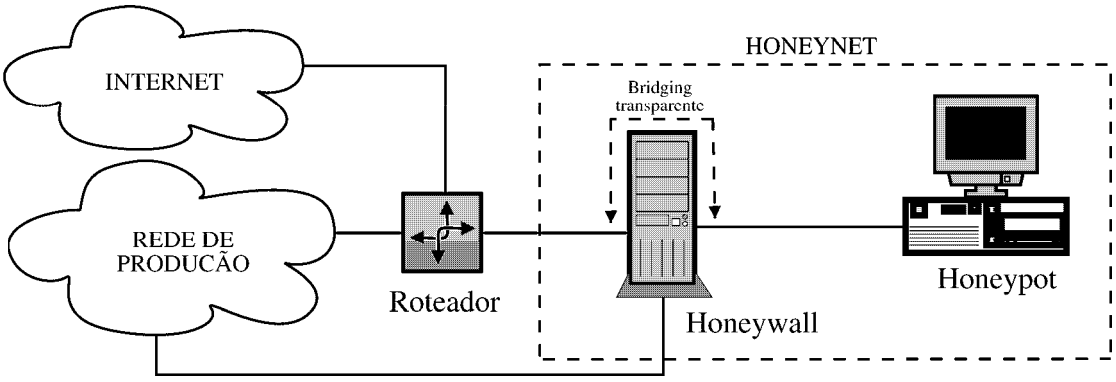

Figura 1: Topologia da *honeynet*

Este artigo descreve um mecanismo para a coleta automática, dinâmica e transparente de evidências digitais em sistemas de arquivos de *honeypots*, eliminando as deficiências encontradas nos métodos tradicionais.

Inicialmente, na Seção 2, será descrito o ambiente de rede no qual o *honeypot* está implantado. A Seção 3 descreverá o mecanismo de coleta de evidências e a Seção 4 analisará os resultados experimentais obtidos com um protótipo do mecanismo em um caso real de intrusão. Finalmente, a Seção 5 concluirá o artigo com as considerações finais sobre o trabalho, e sugestões de possíveis extensões e melhorias.

2. O Ambiente

A escolha do ambiente onde residirá um *honeypot* é uma etapa importante na sua implantação, pois influencia decisivamente a natureza, quantidade e qualidade dos dados coletados [Spitzner, 2002]. O *honeypot* estudado neste artigo reside em uma *honeynet* do tipo *GenII* (2^a geração) [The Honeynet Project, 2003a], como aquela ilustrada na Figura 1.

Uma *honeynet* pode ser definida como um ambiente de rede bem delimitado e altamente controlado, onde residem um ou mais *honeypots* de alta interatividade [Spitzner, 2000]. *Honeynets GenII* consistem em um segmento de rede isolado do ambiente de produção, onde há uma máquina denominada *honeywall* que atua como mediadora do tráfego que entra e sai do *honeypot*.

Essa máquina atua basicamente como uma *firewall* "às avessas", capturando e analisando o tráfego vindo de fora (tráfego malicioso, em princípio), e controlando ativamente o tráfego originado no *honeypot* e dirigido para fora (gerado pela atividade de um invasor, em princípio).

A monitoração do tráfego entrante é feita através do capturador de tráfego *Tcpdump* e do sistema de detecção de intrusão *Snort*. Este último analisa o tráfego em busca de assinaturas de ataques conhecidos, com o objetivo de alertar o administrador para a ocorrência de ataques ao *honeypot*. O controle do tráfego sainte, por outro lado, é necessário para que um invasor não utilize o honeypot como base de lançamento de ataques para outras máquinas situadas fora do perímetro da rede. Este controle faz uso de várias técnicas que estão fora do escopo deste artigo (maiores detalhes em [Steding-Jessen et al., 2003]).

O paradigma *GenII* se utiliza de outra técnica interessante: o *bridging* transparente [Tanenbaum, 2003], ativado na *honeywall*. Com esta técnica, a *honeywall* atua repassando os quadros de rede de um extremo ao outro da *honeynet* (Figura 1) sem realizar qualquer alteração em seu conteúdo. Esta transparência a torna praticamente indetectável por invasores, uma qualidade importantíssima em qualquer solução de *honeypots*.

Por último, vale observar que a *honeywall* possui uma terceira interface de rede, conectada à rede de produção, cuja finalidade é facilitar a sua administração e o acesso aos dados coletados. Esta configuração não quebra o requisito de isolamento da *honeynet*, pois as suas outras duas interfaces de rede não possuem endereços IP associados (justamente por operarem em modo *bridging transparente*), eliminando o risco de que um intruso a invada e comprometa a segurança da rede de produção.

3. Descrição do Mecanismo

O mecanismo de coleta de evidências proposto neste artigo consiste em uma aplicação distribuída composta por dois módulos principais: um *módulo interceptador*, que opera no *honeypot*; e um *módulo receptor*, que opera na *honeywall*.

A coleta de evidências propriamente dita é feita pelo módulo interceptador, que atua em espaço de kernel interceptando dinamicamente as chamadas de sistema que realizam escrita no sistema de arquivos do *honeypot*, e enviando pela rede os dados de contexto de cada chamada interceptada a uma máquina segura à qual o intruso não tenha acesso: a *honeywall*. Este envio é feito de maneira totalmente oculta a um invasor que esteja executando um capturador de pacotes no *honeypot*, pois não interage com o subsistema de rede do kernel, se comunicando diretamente com o *driver* da interface de rede (maiores detalhes na Seção 3.2.4).

Na *honeywall*, os dados enviados pelo módulo interceptador são capturados de forma totalmente passiva (já que a *honeywall* opera em modo *bridging transparente*) pelo módulo receptor, e registrados em um arquivo local. Este arquivo é utilizado então como entrada a um *processador de evidências*, que recupera os dados de contexto de cada chamada de sistema interceptada no *honeypot* e as reproduz localmente, de forma seqüencial. Como resultado desta reprodução, é gerada automaticamente, no sistema de arquivos da *honeywall*, uma árvore de diretórios e arquivos contendo todas as evidências deixadas por um intruso no sistema de arquivos do *honeypot*, desde o início da invasão, até o seu término.

As seções seguintes detalharão o funcionamento dos módulos interceptador e receptor, apresentados acima, após uma breve introdução ao funcionamento de chamadas de sistema no Linux.

3.1. Chamadas de Sistema

Os processadores modernos trabalham em dois modos distintos: o modo usuário e o modo supervisor [Silberschatz et al., 2002]. No primeiro, o processador sofre restrições de acesso e se encontra impossibilitado de executar certas instruções privilegiadas que acessam diretamente o hardware da máquina, de forma a impedir que algum código malicioso (ou mal programado) provoque danos no sistema. Já em modo supervisor, o processador está livre de qualquer restrição, podendo executar todas as instruções e acessar diretamente o hardware.

Nos sistemas operacionais modernos, o *kernel* é executado em modo supervisor, enquanto os processos comuns são executados em modo usuário. Assim, é necessária a existência de uma *API* (*Application Programming Interface*) entre ambos, de modo que os processos possam requisitar ao *kernel* a realização de tarefas privilegiadas, como a escrita de um arquivo ao disco, ou o envio de um pacote pela rede. As *chamadas de sistema* provêem essa interface, realizando a transição entre os dois modos, e possibilitando que o *kernel* satisfaça as necessidades dos processos [Silberschatz et al., 2002].

Já foi observado que todo e qualquer acesso ao hardware por parte de um processo de usuário deve necessariamente invocar uma chamada de sistema, pois somente o código do *kernel*

possui o privilégio necessário para tal. Este fato torna a monitoração de chamadas de sistema ligadas a alterações no sistema de arquivos uma técnica bastante atraente para os objetivos deste trabalho, além de possuir a vantagem de estar protegida contra a ação direta do invasor. Esta técnica será discutida em detalhes na Seção 3.2.

3.2. Módulo Interceptador

O módulo interceptador, que opera no próprio *honeypot*, foi implementado como um *LKM* (*Linux Kernel Module*) [Bovet and Cesati, 2002], isto é, um código-objeto carregado dinamicamente em espaço de *kernel*. Isto é necessário para que o módulo consiga realizar alterações nas estruturas de dados do *kernel*, entre as quais se encontra a tabela de chamadas de sistema.

Uma vez que o *LKM* esteja carregado no *kernel* do *honeypot*, torna-se necessário garantir a sua invisibilidade para que não seja detectado por invasores. Em termos práticos, isso equivale a fazer com que o mesmo não apareça na listagem exibida pelo comando *lsmod*. Esta medida pode ser implementada através do carregamento de um outro *LKM*, que remove o nó referente ao módulo interceptador da lista ligada de módulos mantida na memória do *kernel*, fazendo com que não seja exibido na saída do comando.

O módulo interceptador consiste em uma extensão daquele implementado no *Sebek* [The Honeynet Project, 2003b], uma ferramenta utilizada em *honeypots* que monitora a chamada de sistema *sys_read* a fim de contornar a proteção criptográfica oferecida pelas ferramentas *SSH* e *SCP*, muitas vezes utilizadas por atacantes para cifrar sua comunicação com a máquina invadida. O módulo interceptador estende o *Sebek* no sentido em que monitora não apenas uma, mas todas as principais chamadas de sistema envolvidas diretamente ou indiretamente na realização de alterações ao sistema de arquivos. Estas se encontram listadas abaixo:

- Criação de *hard-links*: *sys_link()*;
- Criação de links simbólicos: *sys_symlink()*;
- Criação e abertura de arquivos: *sys_open(), sys_creat()*;
- Criação de diretórios: *sys_mkdir()*;
- Escrita em arquivos: *sys_write(), sys_pwrite()*;
- Mapeamento de arquivos em memória: *sys_mmap()*;
- Fechamento de arquivos: *sys_close()*;
- Remoção de arquivos: *sys_unlink()*;
- Remoção de diretórios: *sys_rmdir()*;
- Renomeação de arquivos e diretórios: *sys_rename()*;
- Truncamento de arquivos: *sys_truncate(), sys_ftruncate()*;
- Gerenciamento de permissões: *sys_chmod(), sys_fchmod()*;
- Gerenciamento de posse: *sys_chown(), sys_lchown(), sys_chgrp()*.

Por motivos relacionados à simplicidade, no protótipo foram implementados interceptadores somente para as chamadas *sys_open*, *sys_close*, *sys_write*, *sys_rename* e *sys_mkdir*. No caso de *sys_open*, foi tratado apenas o ato de criação de arquivos, deixando de lado o caso em que há a abertura de um arquivo pré-existente.

O funcionamento do módulo pode ser dividido em quatro etapas seqüenciais, a serem detalhadas nas próximas seções. Será utilizada a função interceptadora da chamada *sys_open* (abertura e criação de novos arquivos) como exemplo de implementação.

3.2.1. Etapa 1:Instalação dos interceptadores

A interceptação de chamadas de sistema é uma técnica bastante conhecida e amplamente utilizada [Provos, 2002, Jain and Sekar, 2000]. Esta técnica consiste na alteração dos apontadores armazena-

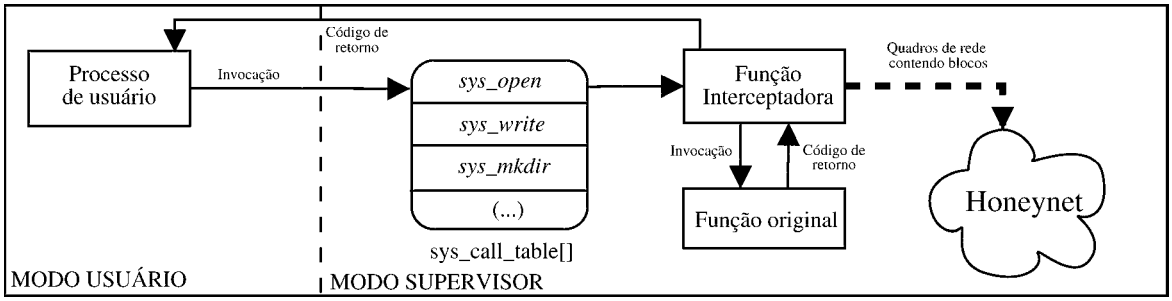

Figura 2: Interceptação da chamada *sys_open*

dos na tabela de chamadas de sistema (residente na memória do *kernel*), de forma a fazer com que determinadas chamadas executem, antes da função original, uma função interceptadora (Figura 2). A implementação desta técnica para a chamada *sys_open* é ilustrado abaixo:

```
int (*old_open)(const char *pathname, int flags, mode_t mode);
asmlinkage int new_open(const char *pathname, int flags, mode_t mode);

asmlinkage int new_open(const char *pathname, int flags, mode_t mode){ [...] }

int init_module(void){
  (unsigned long *)old_open = sys_call_table[__NR_open];
  sys_call_table[__NR_open] = (unsigned long *)new_open;
}
```

No trecho de código acima, vê-se que na função principal do módulo (*init_module()*), a entrada referente à chamada *sys_open* na tabela de chamadas de sistema (referenciada pelo vetor *sys_call_table[]*) é sobrescrita com o endereço da função interceptadora *new_open*. Antes disso, porém, o valor antigo armazenado nessa posição é atribuído à variável *old_open*, a ser utilizada na invocação da função original.

3.2.2. Etapa 2: Invocação da chamada original e checagens relacionadas

Nas funções interceptadoras, a primeira tarefa desempenhada consiste na invocação da função original com os parâmetros passados pelo usuário, como pode ser visto abaixo:

```
  res = old_open(pathname, flags, mode);
  if((res != -1) && ((flags & O_CREAT) != 0)){
    // [...]
  }
  return res;
}
```

Essa invocação prévia é necessária para que saibamos de antemão se a função original será executada com sucesso no *honeypot*, ou retornará algum erro. Caso haja sucesso, o restante do código interceptador é executado; caso contrário, a função interceptadora é encerrada e o valor armazenado em *res* é retornado ao processo que invocou a chamada (Figura 2). Outra checagem feita no código acima diz respeito às *flags* de abertura do arquivo. O corpo principal da função interceptadora é executado somente se o arquivo em questão estiver sendo criado (`O_CREAT`), pois, como já foi citado anteriormente, deseja-se tratar somente este tipo de evento no protótipo, ignorando a abertura de arquivos pré-existentes. Para as outras chamadas de sistema, outras checagens poderão ser feitas.

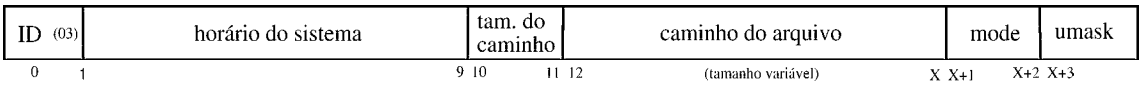

Figura 3: Estrutura do bloco referente à chamada *sys_open*

3.2.3. Etapa 3: Obtenção de Dados Adicionais e Montagem do Bloco

Nesta etapa, é realizada a obtenção de todos os dados que serão necessários para se compor uma unidade lógica (referida a partir deste ponto como *bloco*) a ser remetida à rede. Um bloco deve possuir todos os dados necessários para a reprodução de uma chamada pelo módulo receptor na *honeywall*. Dessa forma, além dos parâmetro passados à chamada de sistema, deverão ser obtidos outros dados de contexto do processo em execução e do sistema como um todo.

Para a chamada *sys_open*, em particular, estes dados incluem o caminho corrente do processo atual (a ser utilizado na composição do caminho absoluto do arquivo), horário do sistema, a máscara de permissões do processo, o nome do arquivo passado como parâmetro e as permissões de criação do mesmo. Outro dado importante é o identificador da chamada em questão. Todas as chamadas interceptadas possuem um inteiro associado que as identificam univocamente em um determinado bloco. Este identificador é utilizado pelo módulo receptor para identificar o tipo de chamada de sistema à qual um bloco corresponde, a fim de interpretá-lo da maneira correta. Para os blocos referentes à chamada *sys_open*, por exemplo, foi convencionado o identificador inteiro **03**, e assim analogamente para as outras chamadas. Para as outras chamadas que não *sys_open*, outros dados podem ser recuperados, dependendo da tarefa executada pela chamada em questão e dos parâmetros que recebe.

Em seguida, monta-se o bloco que será remetido ao módulo receptor. No caso da chamada *sys_open*, o bloco criado possui o formato ilustrado na Figura 3.

3.2.4. Etapa 4: Criação e Transmissão dos Quadros Ethernet

Após a criação dos blocos, estes são encapsulados em quadros Ethernet e enviados à rede. No envio, são utilizados os protocolos Ethernet, IP e UDP como protocolos de nível de enlace, rede e transporte, respectivamente.

A geração dos quadros é feita através da função *gen_pkt*, utilizada no *Sebek* para o mesmo propósito. Esta função inicialmente aloca um *socket buffer* [Bovet and Cesati, 2002] (estrutura do kernel utilizada para o armazenamento temporário de quadros de rede), e compõe os cabeçalhos UDP, IP e Ethernet. Também é feita a cópia do bloco para a área de *payload* do *socket buffer*. A princípio, o conteúdo dos campos dos cabeçalhos Ethernet, IP e UDP é irrelevante, já que as interfaces de rede da *honeywall* não possuem endereços IP, e os quadros devem ser capturados de forma passiva. Deve haver, no entanto, algum método para identificar univocamente os quadros gerados pelo módulo interceptador, a fim de que o módulo receptor possa identificá-los em meio ao universo de quadros que trafegam pela *honeynet*. Neste trabalho, seguindo o exemplo do *Sebek*, foi escolhido o campo *Porta Destino* do cabeçalho UDP para este fim, com o valor *1666*.

Via de regra, há uma relação biunívoca entre blocos e quadros de rede, mas há uma exceção: a chamada *sys_write*. Como, por vezes, esta chamada manipula grandes quantidades de dados, torna-se necessária a divisão do bloco em quadros menores (convencionou-se 4 KB). Após a criação do quadro Ethernet, resta apenas a tarefa de enviá-lo à rede, transferindo-o ao *driver* de dispositivo apropriado. Este envio é feito através da invocação da função de kernel *hard_start_xmit*.

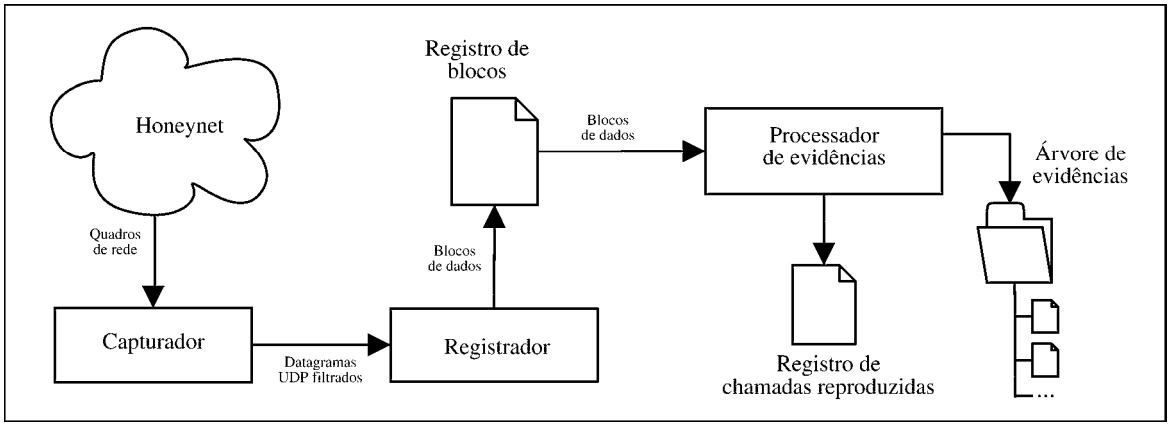

Figura 4: Módulo Receptor

Por ser auto-suficiente na geração dos quadros Ethernet, e comunicar-se diretamente com o *driver* de dispositivo, o mecanismo de envio contorna todos os ganchos de escuta existentes no subsistema de rede do kernel (*netfilter*). Com isso, consegue-se que o envio dos dados seja invisível a um invasor que esteja executando um capturador de pacotes (como o *tcpdump*) no *honeypot*.

3.3. Módulo Receptor

O módulo receptor, cujo funcionamento é ilustrado na Figura 4, opera na *honeywall* como uma aplicação de usuário. Ao contrário do interceptador, o módulo receptor é executado em uma máquina segura, e portanto não há qualquer necessidade de se tomar medidas para ocultar a sua presença.

O seu funcionamento consiste na captura passiva dos datagramas UDP que trafegam pela *honeynet* seguida pela checagem da porta UDP destino, a fim de identificar aqueles enviados pelo módulo interceptador. A partir dos datagramas identificados, os blocos são extraídos e gravados seqüencialmente em um *registro de blocos*. Este registro contém nada mais nada menos do que a cronologia completa das alterações feitas no sistema de arquivo do *honeypot*, desde a ativação do mecanismo, até o instante mais recente. Com esses dados em mãos, abre-se uma miríade de possibilidades no tocante aos tipos de reconstituição de evidências que podem ser realizadas. Pode-se desejar, por exemplo, obter o conjunto de todos os arquivos criados pelo intruso no decorrer da invasão. Ou, quem sabe, gerar um sumário contendo todos os arquivos alterados pelo intruso, com o conteúdo da alteração. Há possibilidades ainda mais instigantes: a alta granularidade temporal dos dados coletados permite, por exemplo, que se crie uma "linha do tempo", contendo a evolução completa de um conjunto de arquivos (e por que não, do sistema de arquivos como um todo) do *honeypot*. Para esta última possibilidade, no entanto, torna-se necessária a presença de uma cópia local do sistema de arquivos original do *honeypot* (antes de ter sofrido alterações provocadas por invasores), a ser utilizada como estado inicial na reconstituição das evidências.

Em todos os casos citados acima, a reconstituição de evidências é feita por um *processador de evidências*, que integra o módulo receptor. Dependendo do tipo de reconstituição, seu funcionamento varia, mas em todos os casos, a cronologia básica de eventos é a mesma: é feita uma análise seqüencial dos blocos recebidos, interpretando o seu conteúdo e reproduzindo localmente a chamada de sistema representada por cada bloco. Esta reprodução é feita com base no campo identificador de cada bloco, utilizando-o para determinar o tipo de chamada de sistema à qual o bloco se refere. Em seguida, os outros campos do bloco são passados a uma função específica, e a chamada em questão é executada.

Todas as evidências reconstituídas pelo processador de evidências são confinadas a uma

hierarquia de diretórios (espelhando a hierarquia do sistema de arquivos do *honeypot*), cuja raiz é especificada na configuração do módulo receptor. Esse diretório-raiz passa a conter, portanto, uma *árvore de evidências*, contendo todas as evidências (arquivos e diretórios) criadas pelo intruso no *honeypot* e reconstituídas na *honeywall*, pelo processador de evidências.

Além da árvore de evidências, o processador de evidências também gera um registro de todas as chamadas reproduzidas. A importância deste registro se torna clara ao considerarmos o caso em que, por exemplo, há remoção de arquivos ou diretórios. Estes ítens não constariam da árvore de evidências e, sem este registro, não tomaríamos conhecimento da remoção.

Da mesma forma que no módulo interceptador, no módulo receptor do protótipo foi implementado somente o tratamento das chamadas *sys_open*, *sys_close*, *sys_write*, *sys_rename* e *sys_mkdir*, Sendo assim, o protótipo trabalha apenas com a reconstituição da criação e renomeação de novos arquivos e diretórios, deixando de lado outros tipos de reconstituições mais complexas.

4. Resultados Experimentais

Como etapa final deste trabalho, o protótipo implementado foi submetido a um processo de validação empírica, através da análise de seu comportamento quando submetido a casos reais de intrusão. Para este fim, os módulos interceptador e receptor do protótipo foram devidamente instalados e acionados no *honeypot* (executando kernel 2.4.19) e na *honeywall* (idem), respectivamente, estando estes devidamente confinados em uma *honeynet* como aquela descrita na Seção 2. Também foi instalado no *honeypot* um registrador de comandos, a fim de se conhecer os comandos digitados pelos invasores no *shell* do *honeypot*. Este registrador opera de forma muito similar ao módulo interceptador, capturando em espaço de kernel os comandos digitados por intrusos e enviando-os pela rede de maneira oculta.

Após 20 dias em operação, o *honeypot* foi atacado e comprometido. O atacante aproveitou-se de uma vulnerabilidade remotamente explorável no servidor FTP do *honeypot* para obter um *prompt* de shell com privilégios de super-usuário. A ocorrência do ataque foi imediatamente notificada pelo *IDS Snort*:

```
1/04-17:24:49.684235  [**] [1:1630:5] FTP EXPLOIT CWD overflow [**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
```

Poucos segundos após o ataque, o capturador de pacotes carregado na *honeywall* começou a notificar a captura de datagramas UDP originados do registrador de comandos, contendo os comandos digitados pelo invasor no *honeypot*. A listagem dos comandos executados segue abaixo:

```
# unset $HISTFILE; id; ls; wget; wget ftp://whitez:whiter0x@217.215.111.13/www/ \
vmkit2k.tar.gz; tar xvzf vmkit2k.tar.gz; cd vmkit2k; ./install spike68
```

O intruso começou removendo a variável de ambiente $HISTFILE, na esperança de que seus comandos não fossem registrados pelo shell. Em seguida, executou o comando *id*, confirmando que possuía privilégios de superusuário. Ele então utilizou a ferramenta *wget* para transferir via FTP o arquivo `vmkit2k.tar.gz` de um servidor remoto, descompactou-o, entrou no diretório criado e executou o programa *install*.

Após esta série de comandos, o intruso tentou se conectar à porta TCP 15000, onde presumivelmente instalara uma *backdoor*. Após verificar que estava funcionando da maneira esperada, ele se desconectou do *honeypot*.

Tão logo o intruso iniciou a transferência do arquivo *vmkit2k.tar.gz*, o módulo receptor começou a registrar a captura de quadros de rede originados do módulo interceptador do *honeypot*. O primeiro quadro recebido segue abaixo:

Tabela 1: Histograma dos blocos recebidos durante a invasão

Chamada de sistema	Blocos recebidos	%	Bytes recebidos	%
sys_write	6792	96,38%	6757481	99,68%
sys_open	86	1,22%	11192	0,16%
sys_close	86	1,22%	9314	0,14%
sys_rename	22	0,31%	1152	0,02%
sys_mkdir	12	0,17%	337	<0,01%
Total	7047	100%	6779476	100%

```
0000  00 02 b3 00 00 00 01 00  00 0f c4 a9 08 00 45 0d   ..............E.
0010  00 49 00 50 00 00 01 11  21 2c 8f 6a 3c a2 8f 6a   .I.P....!,.j<..j
0020  3c a5 06 82 06 82 00 35  29 06 03 00 87 fd a7 3f   <.......5).......?
0030  11 b6 09 00  17 00 2f 76 61 72 2f 66 74 70 2f 76   ....../var/ftp/v
0040  6d 6b 69 74  32 6b 2e 74 61 72 2e 67 7a b6 01 12   mkit2k.tar.gz....
0050  00 00 00                                           ...
```

Excluindo-se os 42 bytes iniciais referentes aos cabeçalhos Ethernet, IP e UDP, nota-se que o primeiro byte do *payload* é **03**. Este byte representa o campo *id* do bloco encapsulado; no caso, referente à chamada *sys_open* (Figura 3). Como o módulo interceptador implementado no protótipo trata somente chamadas *sys_open* relacionadas à criação de arquivos, logo podemos concluir que o quadro de rede ilustrado acima contém um bloco referente à criação do arquivo `/var/ftp/vmkit2k.tar.gz` no *honeypot*.

Após este bloco inicial, se seguiram muitos outros, referentes à criação, escrita e renomeação de arquivos e diretórios. As estatísticas do recebimento de blocos por parte do módulo receptor estão registradas na Tabela 1.

Os blocos recebidos foram gravados seqüencialmente no registro de blocos, totalizando 6,46 MB de dados. Tomando-os como entrada, o processador de evidências gerou uma árvore de evidências no diretório `/EVIDENCIAS` do sistema de arquivos da *honeywall*. Esta árvore contém todos os diretórios e arquivos criados pelo intruso no *honeypot*, dispostos de forma a espelhar o sistema de arquivos do mesmo. Isso significa que o diretório `/EVIDENCIAS/var/ftp` da *honeywall* contém as evidências criadas no diretório `/var/ftp` do *honeypot*:

```
root@honeywall / $ ls -l /EVIDENCIAS/var/ftp
total 1444
drwxr-xr-x    4 root root        4096 Nov  7 20:09 vmkit2k
-rw-r--r--    1 root root     1467505 Nov  7 20:09 vmkit2k.tar.gz
```

Conforme o esperado, o arquivo `vmkit2k.tar.gz` transferido via FTP pelo intruso foi corretamente reconstituído. Sua integridade (assim como a de todos os outros arquivos reconstituídos) foi confirmada através da comparação com os resumos MD5 dos arquivos originais (do *honeypot*).

Vários outros arquivos e diretórios (que não serão exibidos aqui por motivos de espaço) foram reconstituídos pelo módulo processador, fornecendo dados bastante interessantes sobre o programa instalado pelo intruso. Nos diretórios `/usr/bin`, `/usr/sbin` e `/sbin` do *honeypot*, por exemplo, constatou-se que vários binários do sistema, como `netstat`, `sshd`, `ps`, `ifconfig`, `syslogd`, entre outros, foram substituídos com novos binários de mesmo nome, sugerindo tratar-se de um *rootkit*. A análise das evidências que se seguiu confirmou esta hipótese.

Outro aspecto do funcionamento do mecanismo que merece consideração diz respeito ao impacto no desempenho geral do *honeypot* criado pela presença do módulo interceptador. É natural que surja um *overhead* quando se faz interceptação de chamadas de sistema, devido ao código

adicional que deve ser executado em cada invocação. Num *honeypot*, porém, esse *overhead* deve ser mantido sob controle, a fim de que o intruso não o perceba e crie suspeitas.

Uma análise da Tabela 1 revela que a chamada *sys_write* constitui o principal gargalo do mecanismo, dada sua fração no número total de blocos (96,38%) e dados (99,68%) recebidos. Este *overhead* foi medido através de experimentos envolvendo a cópia de arquivos (procedimento que utiliza intensamente a chamada *sys_write*) de diferentes tamanhos (de 100KB a 10 MB). Os resultados mostraram que, dependendo do tamanho do arquivo copiado, o aumento no tempo de cópia varia entre 70% e 90%. Estes números, ao contrário do que aparentam, são bastante aceitáveis se for considerado que, para arquivos pequenos e médios (<10MB), nos discos atuais, a cópia não demora mais que algumas frações de segundo. Este *overhead* se torna um problema apenas na gravação de quantidades muito grandes de dados (dezenas ou centenas de MB), ocasionando um atraso perceptível.

5. Conclusão

Neste trabalho foi descrito um mecanismo baseado na interceptação de chamadas de sistemas para a coleta de evidências digitais em sistemas de arquivos de *honeypots*. Este mecanismo elimina as principais deficiências inerentes aos métodos tradicionais, provendo automatização, dinamismo e transparência na coleta de evidências.

No experimento descrito na Seção 4, o protótipo comportou-se da maneira esperada, gerando uma árvore com todos os arquivos e diretórios criados no sistema de arquivos do *honeypot* ao longo da invasão. Este processo se deu de forma totalmente automática, evitando que um tempo valioso fosse gasto na realização de uma análise forense complexa que talvez não trouxesse resultados tão bons. A árvore de evidências gerada foi submetida a um processo de análise de evidências, no qual se determinou que a ferramenta instalada pelo intruso se tratava de um *rootkit*.

O potencial desta abordagem, no entanto, se estende para muito além do que foi implementado no protótipo. O dinamismo e a alta granularidade temporal e informacional com que a coleta de evidências é feita permitem, por exemplo, a reconstituição da árvore de evidências em diferentes instantes de tempo, possibilitando a criação de uma cronologia completa das evidências criadas no decorrer de uma invasão. Assim, mesmo que um invasor seja cuidadoso o bastante para apagar todas as evidências antes de se desconectar do *honeypot*, o módulo processador poderá reconstituí-las, bastando para isso que retroceda na cronologia dos blocos recebidos e reproduza as chamadas de sistema referentes à criação destas evidências. A possibilidade de se trabalhar com a dimensão temporal proporciona uma visão única do *modus operandi* dos invasores, que nem sempre pode ser conseguida com o uso das metodologias estáticas.

Também merece destaque a transparência do mecanismo, realizando a interceptação dos dados, envio dos blocos e reconstituição das evidências de modo totalmente oculto ao intruso. O único revés encontrado neste quesito diz respeito ao impacto de desempenho criado no *honeypot* pela presença do módulo interceptador que, em casos específicos, pode levantar suspeitas por parte do intruso. Entretanto, constatou-se através de medições que este impacto só é perceptível na escrita de volumes muito grandes de dados. Não obstante, a otimização do código interceptador da chamada *sys_write* constitui uma importante extensão deste projeto.

Outra limitação do mecanismo diz respeito à manutenção da integridade da tabela de chamadas de sistema. É fato conhecido que um intruso pode modificá-la através da instalação de *rootkits* baseados em *LKM*, por exemplo, o que poderia resultar na neutralização do módulo interceptador. Outra possível extensão ao mecanismo seria, portanto, a criação de um módulo supervisor que, de algum modo, mantivesse a integridade desta tabela. Com o atual paradigma de segurança,

entretanto, é impossível construir uma solução 100% robusta, já que o invasor geralmente possui privilégios de *root* no *honeypot*, o que se traduz em poderes ilimitados. Com esses poderes, o invasor pode derrotar qualquer sistema de vigilância; trocando o *kernel* da máquina, ou mesmo apagando o sistema de arquivos, entre outras formas. Nada pode ser feito para impedi-lo. Os esforços devem ser concentrados na criação de módulos de vigilância mais transparentes, que não levantem suspeitas. Nesse sentido, a utilização de máquinas virtuais representa um grande avanço, e a adaptação do mecanismo aqui descrito para um gerenciador de máquinas virtuais, como o *User Mode Linux*[1], constitui outra possível extensão deste trabalho.

Referências

Bovet, D. P. and Cesati, M. (2002). *Understanding the Linux Kernel*. O'Reilly, Sebastopol, CA, USA, 2. edition.

Cheswick, W. R. (1992). An evening with berferd in which a cracker is lured, endured and studied. In *Proceedings of the Winter 1992 USENIX Conference*.

dos Reis, M. A. (2003). Forense computacional e sua aplicação em segurança imunológica. Master's thesis, Instituto de Computação - UNICAMP, Campinas, SP.

Equipe Honeynet.BR (2002). Honeynet.BR: Desenvolvimento e Implantação de um Sistema para Avaliação de Atividades Hostis na Internet Brasileira. In *Anais do IV Simpósio sobre Segurança em Informática (SSI'2002)*, São José dos Campos, SP.

Jain, K. and Sekar, R. (2000). User-level infrastructure for system call interposition: A platform for intrusion detection and confinement. In *Proceedings of Network and Distributed System Security (NDSS 2000)*, San Diego, CA, USA.

Provos, N. (2002). Improving host security with system call policies. Technical Report 02-3, University of Michigan.

Provos, N. (2003). Honeyd: A virtual honeypot daemon. In *10th DFN-CERT Workshop*, Hamburg, Germany.

Silberschatz, A., Galvin, P. B., and Gagne, G. (2002). *Operating System Concepts*. John Wiley & Sons, New York, NW, USA, 6. ed edition.

Spitzner, L. (2000). Learning the tools and the tactics of the enemy with honeynets. In *Proceedings of the 12th Annual Computer Security Incident Handling Conference*, Chicago, IL, USA.

Spitzner, L. (2002). *Honeypots: Tracking Hackers*. Addison-Wesley, Boston, MA, USA.

Spitzner, L. (2003a). Honeypots: Definitions and values. Disponível em World Wide Web (Janeiro de 2004): <http://www.tracking-hackers/papers/honeypots.html>.

Spitzner, L. (2003b). Honeytokens: The other honeypot. Disponível em World Wide Web (Janeiro de 2004): <http://www.securityfocus.com/infocus/1713>.

Steding-Jessen, K., Hoepers, C., and Montes, A. (2003). Mecanismos para Contenção de Tráfego Malicioso de Saída em Honeynets. In *Anais do V Simpósio sobre Segurança em Informática (SSI'2003)*, São José dos Campos, SP.

Stoll, C. (1991). *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage*. Pan Books.

Tanenbaum, A. S. (2003). *Computer Networks*. Prentice Hall, Upper Saddle River, NJ, USA, 4. edition.

The Honeynet Project (2001). *Know Your Enemy: Revealing the Security Tools, Tactics, and Motives of the Blackhat Communit*. Addison-Wesley, Indianapolis, IN, USA.

The Honeynet Project (2003a). Know your enemy: Genii honeynets. Disponível em World Wide Web (Janeiro de 2004): <http://project.honeynet.org/papers/gen2/>.

The Honeynet Project (2003b). Know your enemy: Sebek. Disponível em World Wide Web (Janeiro de 2004): <http://www.honeynet.org/papers/sebek.pdf>.

[1] http://user-mode-linux.sourceforge.net